

序言

LaunchPad 口袋实验平台必须搭配 MSP-EXP430G2 LaunchPad 实验板使用。

口袋实验平台正式配套书籍《从零开启大学生电子设计之路——基于 MSP430 LaunchPad 口袋实验平台》，由北京航空航天大学出版社出版。该书内容涵盖单片机入门知识、编程方法、片内外设知识和口袋实验平台例程的详细讲解。

口袋实验平台的全部例程代码可登陆 www.hpati.com 网站进行下载。其他技术支持信息也将在该网站发布，恕不另行通知。

目 录

1	口袋实验平台概述	1
2	硬件	3
3	软件	6
3.1	开发软件	6
3.2	例程代码	6
3.3	例程演示录像	8
3.4	快速培训演示文档	8
4	实验例程简介	10
4.1	观测 DCO 频率变化	10
4.2	中断按键	12
4.3	基于 PWM 的 LED 调光控制	14
4.4	呼吸灯	16
4.5	定时扫描非阻塞按键	18
4.6	长短键识别	20
4.7	电容触摸按键	22
4.8	电容触摸长短键	24
4.9	超级终端人机交互	25
4.10	SD 卡读写扇区	26
4.11	I2C 扩展 IO	29

4.12	LCD 显示自检.....	31
4.13	拨盘电位器.....	34
4.14	温度传感器采样及显示.....	36
4.15	SPWM 波形合成及采样.....	38
4.16	任意波形发生器 AWG.....	41
4.17	基于 AWG 的音频播放器.....	43
4.18	自校验 DCO.....	45

1 口袋实验平台概述

口袋实验平台包含 MSP-EXP430G2 中配套的 MSP430G2553 单片机全部片内外设实验以及 3 个综合性实验，如表 1 所示。

表 1 实验例程列表

外设	实验例程工程名	对应书籍章节
System Clock	4_KEY_LED_Change_DCO	4.8
GPIO	5_Interrupt_Key_LED	5.6
Timer_A	6_PWM_LED	6.7
WDT	7_1_Breath_Lamp	7.8
	7_2_Timer_Key_LED	7.10
	7_3_Key_Long_Short_Mealy	7.12
	7_4_Key_Long_Short_Moore	7.12
Capactive touch	8_1_TouchPad_LED	8.5
	8_2_TouchPad_Long_Short_Mealy	8.6
USCI_UART	10_UART_KeyBoard	10.8
USCI_SPI	11_SPI_SD	11.5
	13_1_SD_Hard_or_Soft_SPI	13.3.2
USCI_I2C	12_I2C_LED_KEY	12.7
	13_2_LED_KEY_I2C_Hard_or_Soft	13.4
	14_LCD_SelfScan_Hard_or_Soft_I2C	14.8
Comparator_A	17_Slope_ADC	17.6
ADC10	19_ADC10_Temperature	19.4
PWM	20_Sin_Gen_and_Sample	20.5
DAC	21_1_DAC_AWG	21.5
	21_2_TF_Audio	
Self-Calibrate DCO Flash	22_DCO_Calb_Test	22.3

口袋实验平台可以不借助其它测试仪器实现对单片机的内部资源和外设的学习和实验。三个综合实验的录像中，使用了部分仪器来丰富实验效果。

- 1) 在例程“20_Sin_Gen_and_Sample”中可以使用示波器观测 PWM 滤波波形，也可以利用仿真器间接观测波形。
- 2) 例程“21_1_DAC_AWG”中使用了示波器，如无示波器，可用例程“21_2_TF_Audio”替代学习 DAC 的使用。
- 3) 例程“22_DCO_Calb_Test”录像中示波器和频率计的使用是为了验证自校验 DCO 频率的精度，利用单片机自制的频率计一样能满足要求，可自行编程实现频率计功能。

2 硬件

口袋实验室硬件原理框图如图 1 所示，参考图 1 和表 2：

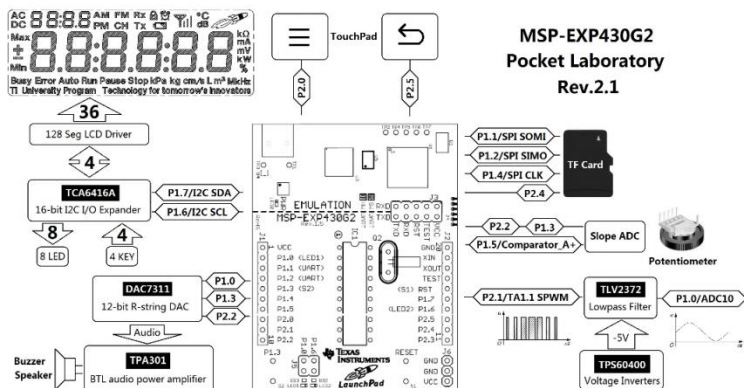


图 1 口袋实验室硬件原理框图

- 1) 显示和输入单元：口袋实验平台利用 I2C 接口的 TCA6416A 扩展出 16 个低速双向 IO (IO00~IO07, IO10~IO17)。4 个扩展 IO 用于控制 LCD 驱动器 HT1621, 4 个用于机械按键输入, 8 个用于 LED 灯柱。
- 2) 触摸按键单元：两个触摸按键占用 P2.0 和 P2.5 两个 GPIO, MSP430G2 系列单片机的 P1 和 P2 全部具备振荡功能。
- 3) 模拟输出单元：外部扩展了 12 位串行数模转换器 DAC7311, 使用 P1.0/P1.3/P2.2 三个普通 GPIO 控制。

这三个 IO 同时被其他单元复用，但由于 DAC7311 都是高阻输入口，所以无需跳线复用。音频功放 TPA301 可以将 DAC 输出进行电流放大，以便驱动喇叭/蜂鸣器负载。

- 4) 扩展存储部分：由 SPI 协议控制 TF 卡，使用 P1.1/SPI SOMI、P1.2/SPI SIMO、P1.4/SPI CLK 三个 USCI 功能 IO 和 1 个普通 P2.4 控制。
- 5) SLOPE ADC 单元：使用拨盘电位器作为待测电阻。P1.5/Comparator_A+、P1.3、P2.2 三个 IO 进行控制。P1.3、P2.2 复用，但同样无需跳线。
- 6) PWM 单元：P2.1/TA1.1 负责输出 SPWM；轨至轨运放 TLV2372 负责将 SPWM 滤波为双极性模拟信号；TPS60400 提供运放所需负电源；三电阻网络负责将双极性信号转变为单极性信号；P1.0/ADC10 负责对单极性信号采样，P1.0 功能复用，同样无需跳线

表 2 口袋实验平台硬件功能单元

名称	元件	IO	功能
IO 扩展	TCA6416A	P1.6/I2C SDA P1.7/I2C SCL	利用 I2C 协议控制 TCA6416A 获得 16 个低速 IO IO00~IO07, IO10~IO17
显示	HT1621 128 段 LCD	IO14/扩展 IO;IO15/扩展 IO IO16/扩展 IO;IO17/扩展 IO	4 个 I2C 扩展 IO 控制 HT1621 HT1621 控制段式液晶
机械 按键	微动开关*4	IO10/扩展 IO;IO11/扩展 IO IO12/扩展 IO;IO13/扩展 IO	4 个 I2C 扩展 IO 识别 机械按键
LED 灯 柱	0603LED*8	IO00/扩展 IO;IO01/扩展 IO IO02/扩展 IO;IO03/扩展 IO IO04/扩展 IO;IO05/扩展 IO IO06/扩展 IO;IO07/扩展 IO	8 个 I2C 扩展 IO 控制 8 个 LED
触摸 按键	覆铜	P2.0 P2.5	电容触摸按键
模拟 输出	DAC7311 TPA301 蜂鸣器	P1.0/复用 P1.3/复用 P2.2/复用	12 位串行 DAC7311 BTL 乙类功率放大器
扩展 存储	TF 卡槽	P1.1/SPI SOMI P1.2/SPI SIMO P1.4/SPI CLK P2.4	基于 SPI 的 TF 卡读写
SLOPE ADC	拨盘电位器	P1.5/Comparator_A+ P1.3/复用 P2.2/复用	利用比较器实现积分型 ADC
PWM	TLV2372 TPS60400 三电阻网络	P2.1/TA1.1 P1.0/ADC10/复用	PWM 波形合成 双极性信号的 ADC 采样

3 软件

口袋实验平台的全部软件资源均可从网络获得。包括单片机开发软件、实验例程代码、实验例程演示录像、带讲解的快速培训演示文档。

3.1 开发软件

口袋实验室使用 TI 公司的 CCS 软件进行开发，下载地址为 <http://www.ti.com.cn/tool/cn/ccstudio>，请选择 CCS5.1 及以上版本。CCS 可永久免费使用代码限制版本。TI MSP430 共建实验室高校可联系 TI 中国大学计划部索取大学计划许可文件来解除代码限制。

有关 CCS 软件的安装使用，请参阅《从零开启大学生电子设计之路—基于 MSP430 LaunchPad 口袋实验平台》或者网络搜索相关教程。唯一提醒是不要有任何中文路径。

3.2 例程代码

从 www.hpati.com 网站上下载例程代码，按工作空间导入 CCS 软件中，可得图2所示的工程导航，共21个实验例程，工程名前的数字代表所属指导书的章节。其中13章的“13_1_SD_Hard_or_Soft_SPI”工程和“13_2_LED_KEY_I2C_

Hard_or_Soft”工程添加了条件编译语句，使用GPIO软件模拟的SPI和I2C协议将“11_SPI_SD”工程和“12_I2C_LED_KEY”工程进行改写。

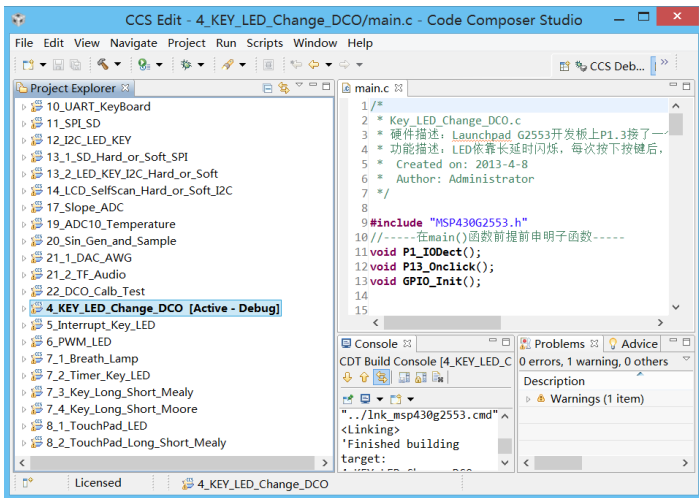



图2 CCS 工程导航

注意：源工程文件是在 CCS5.1 环境下编写，高于 CCS5.1 版本的软件打开工程后工程名会有感叹号，这完全不影响使用。也可按软件提示去自动升级代码，但升级后的代码将无法用低版本 CCS 打开。

3.3 例程演示录像

从www.hpati.com网站上还可以下载例程的演示录像。图3为解压后的实验例程录像文件夹，打开“实验例程录像.ppt”，可观看各个实验例程烧录后的效果。

注意计算机中应安装好相应的视频播放软件，如演示文档不能在页面中播放的现象，请升级至最新的播放器和解码器。再无法解决的，直接点击文件夹中的视频文件播放。

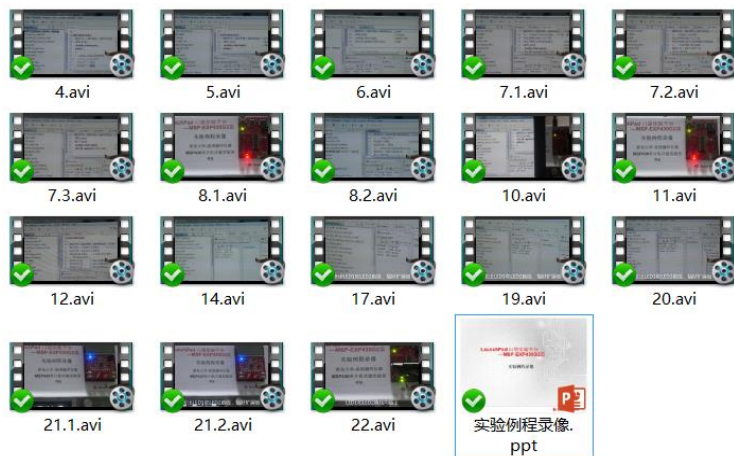


图 3 实验例程文件夹

3.4 快速培训演示文档

从www.hpati.com网站上下载快速培训演示文档，如图4所

示。请使用Microsoft Office PowerPoint 2010以上版本打开，这样就能获得自动播放讲解的功能。

在幻灯片放映菜单中，勾选播放旁白等3个选项，即可获得自动讲解PPT的功能。如无法获得PowerPoint2010以上版本，演示文档也能按普通ppt来使用，可在线观看转换后的视频讲解录像（与自动播放ppt内容一致）。



图 4 带旁白的快速培训 PPT

快速培训演示文档播放时长约 210 分钟，旁白对应的讲解文字已在备注栏显示，如图 5 所示。

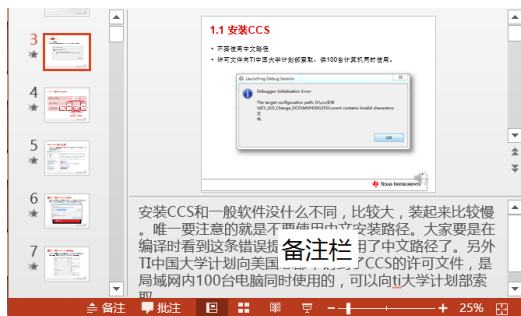


图 5 演示文档的文字备注

4 实验例程简介

本章将简要介绍各实验例程的知识点和注意事项，详细内容请阅读《从零开启大学生电子设计之路—基于 MSP430 LaunchPad 口袋实验平台》。

4.1 观测 DCO 频率变化

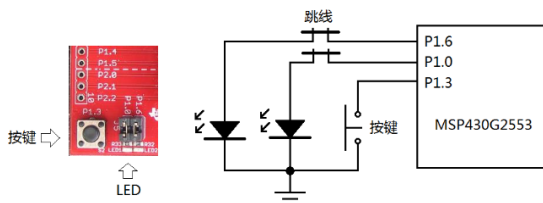


图 6 观测 DCO 频率变化实验原理图

工程名	4_KEY_LED_Change_DCO
实验内容	设定 DCO 频率并观察 DCO 频率变化
实验原理	<ul style="list-style-type: none"> ➤ MSP430G2553 单片机的 CPU 时钟来源于数控振荡器 DCO。 ➤ CPU 通过长延时控制 P1.6 的 LED 闪烁亮灭。 ➤ 单片机识别 P1.3 机械按键以后，改变 DCO 参数，依次设置为 1M, 8M, 12M 和 16MHz，可观察到闪烁频率增加。
注意事项	不使用扩展板进行实验，图 6 所示跳线帽需插上实验。

本例程主要学习单片机时钟的配置方法和前后台的编程思想:

- 1) 主函数main()中, 只调用初始化函数GPIO_Init()和执行后台程序, 本例中后台程序就是长延时改变LED (P1.6) 的亮灭。
- 2) 前后台程序结构中, 前台程序就是各种中断中执行的程序。中断服务子函数PORT1_ISR()中不要去写“实质性”代码, 只调用事件检测函数, 这样能提高程序可读性。
- 3) 事件检测函数P1_IODect(), 在GPIO中断服务子函数中调用, 检测按键是否“一定”被按下, 然后调用事件处理函数。
- 4) 事件处理函数P13_Onclick(), 一旦被调用, 就循环改写DCOCTL和BCSCTL1寄存器, 从而改变DCO (也就是CPU) 频率。
- 5) 前后台程序中, 前台程序的编写是难点。但只要坚持用“事件检测”“事件处理”的方法处理, 就能变得井井有条。

4.2 中断按键

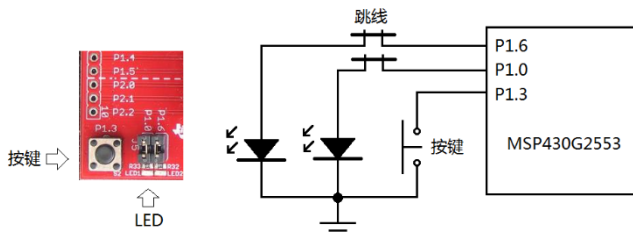


图 7 中断按键实验原理图

工程名	5_Interrupt_Key_LED
实验内容	按键切换两个 LED 的亮灭状态。
实验原理	主函数完成初始化设定后，休眠。 P1.3 事件处理函数中，改写 P1.0 和 P1.6 输出状态。
注意事项	图 7 所示跳线帽需插上实验，机械按键需用程序配置 IO 内部上拉电阻。

本例程主要学习机械按键的延时消抖代码、CPU 休眠与非阻塞编程：

- 1) 中断发生以后，“事件”未必发生，所以才需要进一步调用“事件检测函数”。事件检测函数往往是最难编写的，例如本例中按键延时消抖的处理就是属于事件检测函数。
- 2) 如图8所示，机械按键被按下或松开时，会形成多个干扰脉冲，事件检测函数必须将中断真正的一次按下同0

③④⑤ 区别开。

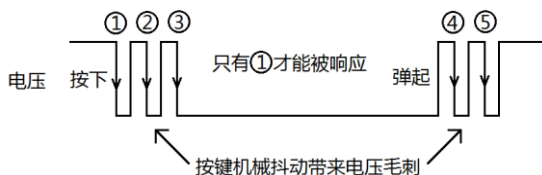


图 8 机械按键的电平抖动

- 3) 事件处理函数包含的意义就是“一旦发生什么事，就干什么”，这就好比是应急响应预案一样，当阅读一个程序的时候，应该最先阅读事件处理函数，这样程序意图就能基本知道。
- 4) 除了读写寄存器（包括改写 IO 输出），数学和逻辑运算外，CPU 干其他事情例如长延时、死循环类型的查询都属于阻塞 CPU 的行为，应坚决避免。
- 5) 当 CPU 确无任务需要执行时，休眠是最佳选择，正确运用各种等级的休眠（可唤醒）可以极大的降低单片机的功耗。

4.3 基于 PWM 的 LED 调光控制

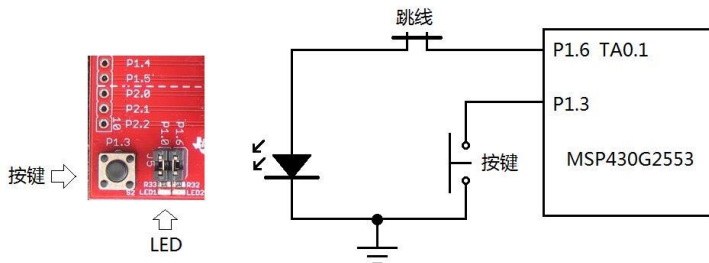


图 9 LED 调光控制实验原理图

工程名	6_PWM_LED
实验内容	通过按键控制 LED 亮度。
实验原理	<ul style="list-style-type: none"> ➤ TA (Timer_A 定时器) 可用于自动输出 PWM 而无需 CPU 干预。 ➤ 编写 PWM 初始化、设定频率、改写占空比的库函数文件 TA_PWM.c。 ➤ 主函数调用库函数 TA0_PWM_Init()完成 TA_PWM 初始化，然后休眠。 ➤ P1.3 事件处理函数中，循环递增静态局部变量 Bright 的值，并调用库函数 TA0_PWM_SetPeriod()改写 PWM 占空比参数。
注意事项	不使用扩展板进行实验，图 9 所示跳线帽需插上实验。

本例程主要学习 TA 自动生成 PWM 的原理，外部库函数文件的使用，静态局部变量的使用。TA 生成 PWM 波形的本质就是通过改写定时器的 TACCR0 寄存器改变 PWM 频率，改写 TACCR1/2 寄存器改变占空比。编写库函数文件方便在各种程序中都能调用。

1) 在 ccs 工程中，库函数文件的路径设置要特别注意。

2) 添加如图 10 所示的

外部文件路径的方法：


在工程上点击右键菜

单，选择最下面的

Properties，依次点击

Build → MSP430

Compiler → Include

Options，点击  添

加文件路径，进入 Add directory path 窗口，依次点击

Workspace → 6_PWM_LED → src → OK 就选定了外部文件路径。

3) 一个好的程序总是尽量使用静态局部变量而避免使用全局变量，这样可以增加程序的可读性。

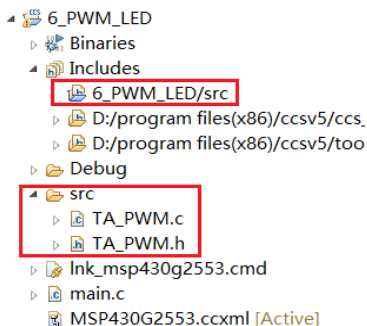


图 10 库函数路径

4.4 呼吸灯

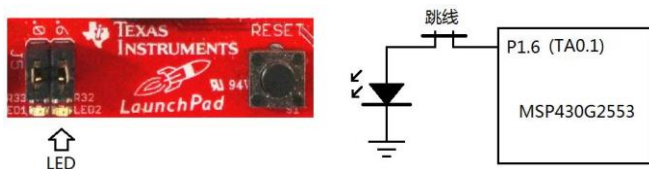


图 11 呼吸灯实验原理图

工程名	7_1_Breath_Lamp
实验内容	通过 WDT 定时周期性调节 LED 亮度，实现呼吸灯效果。
实验原理	<ul style="list-style-type: none"> ➤ WDT 定时器设置为 16ms 中断。 ➤ 在 WDT 中断中，循环递增/递减静态全局变量 Bright，根据 Bright 改写 PWM 占空比。
注意事项	不使用扩展板进行实验，图 11 所示跳线帽需插上实验。

本例程主要学习 WDT 作为节拍定时器的使用，文件系统的使用。

- 1) 节拍定时器在单片机编程中具有重要意义。定时扫描/轮询的方法可以解决大量 CPU 的阻塞问题。
- 2) 节拍定时器不需要定时值连续可变，只要若干常用档位即可，WDT 正好物尽其用。

3) 如图所示，使用文件系统来管理各种函数是大型程序中必须的方法。

4) 除了 TA_PWM 这样的“工具”性质的库函数外，将事件类函数“xxx_Event”，全局变量“xxx_Global”单列文件是十分有必要的。

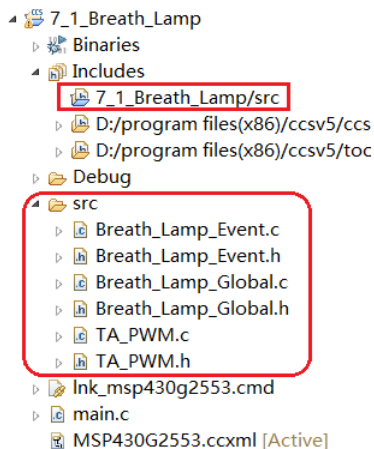


图 12 文件系统

5) 事件函数对于理解程

序意图非常重要，集中放置“事件检测/处理”而不是跟着满世界“乱蹦”的中断去找事件函数，是十分高效的。

6) 全局变量的使用需要非常谨慎，全局变量往往是联系各关键函数的纽带，因此集中放置全局变量也是增加程序可读性的常用做法。

4.5 定时扫描非阻塞按键

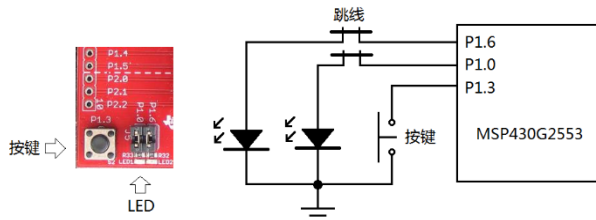


图 13 定时扫描按键实验原理图

工程名	7_2_Timer_Key_LED
实验内容	通过 WDT 定时周期性读取 IO 状态，准确识别按键按下和弹起。
实验原理	<ul style="list-style-type: none"> ➤ WDT 定时器设置为 16ms 中断。 ➤ 在 WDT 中断中，记录下最近两次的 IO 状态存在 KEY_Now 和 KEY_Past 中。根据前高后低可判断按键按下，前低后高可判断按键松开。 ➤ 事件处理函数 P13_Onclick()根据按键状态切换 LED 亮灭。
注意事项	不使用扩展板进行实验，图 13 所示跳线帽需插上实验。

本例程主要学习定时扫描在消除 CPU 阻塞中的作用，以及消抖的原理。

- 1) CPU“无遗漏”地查询事件发生是产生阻塞代码的重要原因。

- 2) 如图 14 所示为定时扫描消抖原理，定时扫描的精髓在于扫描的间隔足够短，保证不会遗漏“事件”。

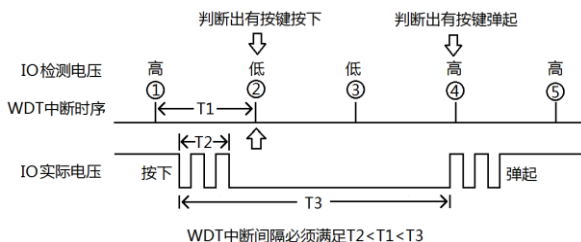


图 14 定时扫描消抖原理图

- 3) 只要 WDT 中断时间比一次按键持续间隔短，就不会漏掉按键。只要比毛刺持续时间长，就不会多检测事件。
- 4) 按键按下的判据为前高后低，按键松开的判据为前低后高。

4.6 长短键识别

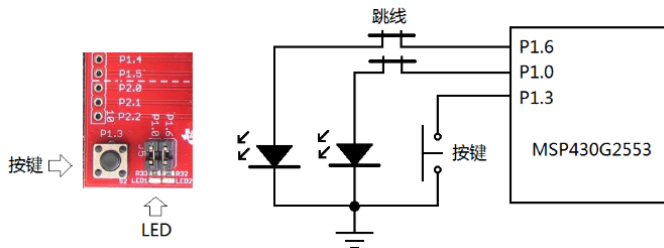


图 15 长短按键实验原理图

工程名	7_3_Key_Long_Short_Mealy; 7_4_Key_Long_Short_Moore
实验内容	通过 WDT 定时周期性读取 IO 状态，识别按键按下和弹起，以及按下的时间。判断为短按键则切换 LED1 亮灭，判断为长按键则切换 LED2 亮灭。
实验原理	<ul style="list-style-type: none"> ➤ WDT 定时器设置为 16ms 中断。在 WDT 中断中，记录下最近两次的 IO 状态存在 KEY_Now 和 KEY_Past 中。根据前高后低可判断按键按下，前低后高可判断按键松开，存入标志位 Key_Dect 中。 ➤ Key_Dect 作为输入量，利用状态机函数 Key_SM()，判断出短按键事件和长按键事件。分别调用各自事件处理函数进行处理。
注意事项	不使用扩展板进行实验，图 15 所示跳线帽需插上实验。

本例程主要学习状态机建模的方法。

- 1) 当事件检测函数不能仅根据当前发生的事就做出最终事件判断时，就需要启用状态机建模的方法了。

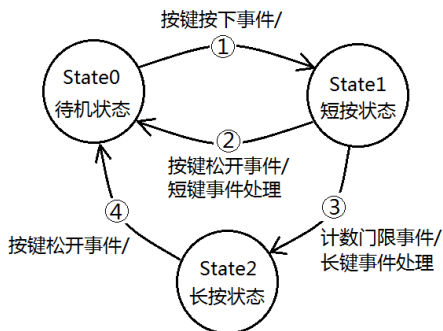


图 16 长短按键的状态机

- 2) 如图 16 所示为状态机的核心，状态转换图。只有当前“状态”加上当前“事件”（输入）才能决定下一步要干什么（下一状态和事件处理）。
- 3) 任何状态机都有两种“公式化”无需动脑的代码描述方法。米利状态机是先 switch(状态)，然后再看发生什么事件。摩尔状态机是先 if(事件)，然后 switch(状态)；两者没有本质区别。本例的两个工程分别用米利状态机和摩尔状态机编写，它们之间只有 Key_SM()函数不一样。图 16 所示状态转换图适用于任何类型的长短按键识别，只要“告诉”状态机按键按下和按键松开两个事件就可以。

4.7 电容触摸按键

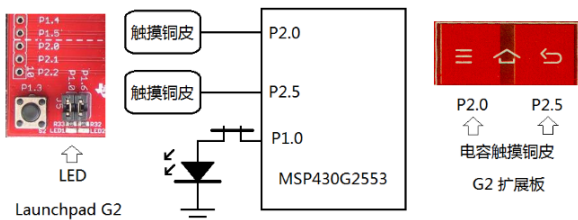


图 17 电容触摸按键实验原理图

工程名	8_1_TouchPad_LED
实验内容	振荡测频法识别触摸按键，根据键值改写 P1.0 口 LED 亮灭。
实验原理	<ul style="list-style-type: none"> ➢ 初始化 GPIO 的振荡功能，并将振荡信号作为 TA 的时钟源。 ➢ 设置 WDT 中断 16ms，16ms 中 TA 定时器的 TAR 计数值即是电容触摸振荡的“频率”，这就是测频原理。频率高于门限，则判断手指接近，记为 1；低于门限，手指离开，记为 0。 ➢ 测频结果存入二维数组 Key_Buff[Key_Num][0/1/2/3]中。Key_Num 用于区分多个触摸按键的序号，0-3 则是最近 4 次的测量结果（FIFO 思想）。 ➢ 调用判据函数 Key_Judge()，只有连续 4 次测频结果为 1，才算“电容触摸”按键真按下，只有连续 4 次测频结果均为 0，才算按键松开。 ➢ 将最终按键的判断结果存入全局变量 TouchIN 中。 ➢ 按需查询 TouchIN，即可作相应事件处理。
注意事项	测频的“家伙”只有一套，所以多个触摸按键需要轮流测频，通过静态局部变量 Key_Num 区分触摸按键编号。中间的“home”按键相当于手指同时触摸到两块铜皮，TouchIN 中 2 位为 1 的情况。

本例程主要学习振荡 IO 的测频方法，FIFO 原理，全局变量的作用。对于带振荡功能的 IO，电容触摸按键识别本质就是测频。测频的方法也很简单，就是在 16ms 的 WDT 中断中数 TA 的计数值。难点在于多个触摸按键怎么处理，触摸判别的可靠性，以及如何“方便”地使用触摸按键，不能总是把它当特别麻烦的特殊按键来看待。

- 1) 所有的工作都在 16ms 的 WDT 中断中进行，用 TouchIN_Dect()函数完成触摸按键的识别，并将结果更新在全局变量 TouchIN 中。也就是说，只要每 16ms 调用一次 TouchIN_Dect()，就能保证全局变量 TouchIN 中存的是按键键值。
- 2) 无论有多少个触摸按键，轮流测频，并记录数据。
- 3) 引入 FIFO 的概念，依靠最近 4 次的测频结果，判断按键是否按下。
- 4) TouchIN 这个全局变量的地位等同于单片机 IO 状态寄存器 PxIN。

4.8 电容触摸长短键

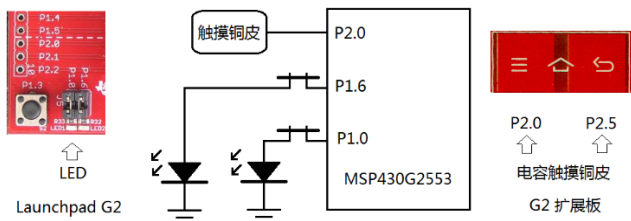


图 18 电容触摸长短键实验原理图

工程名	8_2_TouchPad_Long_Short_Mealy
实验内容	调用电容触摸库函数，使用状态机识别出长短触摸按键，短按键控制 P1.0 口 LED 亮灭，长按键控制 P1.6 口 LED 亮灭。
实验原理	<ul style="list-style-type: none"> ➢ 在 16ms 的 WDT 中断中，调用 TouchIN_Dect()库函数，随时保持 TouchIN 数据准确。 ➢ 把 TouchIN 当做机械按键消抖后的按键结果“Key_Dect”来使用，套入 Key_SM()状态机函数，实现长短按键识别。 ➢ 根据长短按键，调用事件处理函数。
注意事项	按键按下和按键松开的判据为 TouchIN 前 0 后 1 和前 1 后 0，这与机械按键的电平正好是相反的。

本例程主要学习状态机代码的移植，和硬件无关的编程思想。

- 1) WDT 定时调用 TouchIN.c 中的外部函数 TouchIN_Dect() 就可以保证全局变量 TouchIN 存的就是最新的触摸按键键值，该键值是无需再消抖处理的。
- 2) 机械按键的长短键状态机与触摸按键的状态机没有任何区别。参考前面的图 16，状态都是 3 个，按键按下和按

键松开的判据替换成判断 TouchIN 前 0 后 1 和前 1 后 0 即可。

4.9 超级终端人机交互

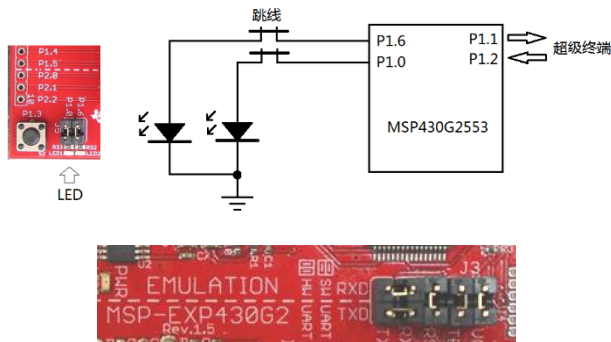


图 19 超级终端实验原理图

工程名	10_UART_KeyBoard
实验内容	利用计算机超级终端控制 G2 板上的两个 LED 亮灭。
实验原理	<ul style="list-style-type: none"> ➢ G2 LaunchPad 支持直接通过 USB 下载口以 UART 方式连接计算机。 ➢ UART 设为波特率 9600, 8 位数据, 无校验, 1 位停止位, 可使用 Grace 配置后再移植。 ➢ 引入软件 FIFO, 分别实现 Tx 和 Rx 数据无阻塞收发。 ➢ CPU 对 Rx 数据进行各种判别, 控制相应 LED 亮灭, 并 Tx 数据回计算机以实现回显和提示。
注意事项	要实现 UART 连接, 必须将图 19 所示的 RXD/TXD 两个跳线横着插。

本例程主要学习 UART 的配置，软件 FIFO 的使用，文件管理。

- 1) UART 的初始化较为复杂，可以使用 Grace 帮助配置寄存器，并单独建一个初始化文件。
- 2) 由于 CPU 读写 UART 缓存的速度极快，而 UART 与计算机通信的速度极慢，所以 CPU 等待过程中容易发生阻塞。
- 3) FIFO 专为解决高低速设备兼容而生。后台程序中 CPU 读写 FIFO，而不是直接读写 UART 缓存。UART 收发中断中，再完成 FIFO 与 UART 缓存间的数据交换。
- 4) 特别注意只要缓存中有数据，UART 是可以“自动连发”的，但是一旦缓存中无数据，下次再要发送是需要手动触发的。

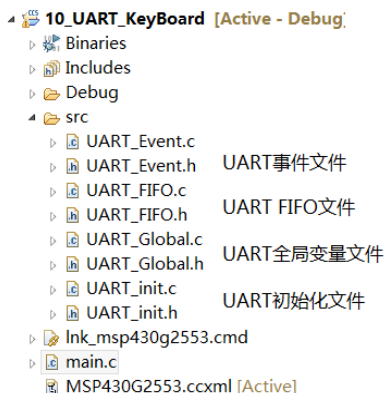


图 20 UART 工程的文件系统

4.10 SD 卡读写扇区

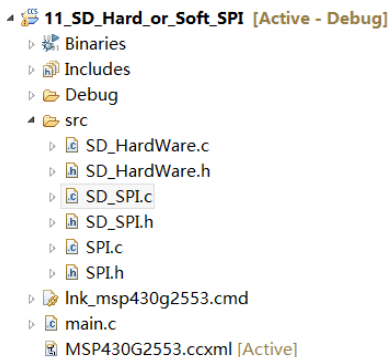
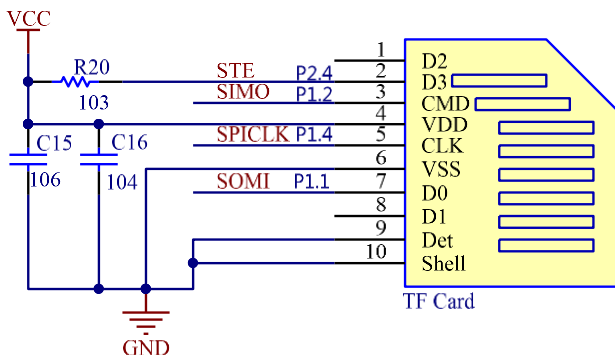


图 21 SD 卡实验原理图及例程文件.

工程名	11_SPI_SD; 13_1_SD_Hard_or_Soft_SPI
实验内容	读写 TF 中指定扇区的数据。
实验原理	<ul style="list-style-type: none"> ➢ 配置 DCO 频率→设置 GPIO 方向→调用库函数 USCI_A0_init()实现 SPI 初始化→调用库函数 SD_Init()实现 SD 卡初始化。 ➢ 调用写 SD 函数 My_Write_Data(), 将 0~127 这 128 个数据写入 SD 指定扇区 “SD_SECTOR_ADDR” ➢ 调用 SD 读函数 My_Read_Data(), 从 SD 卡中指定扇区 “SD_SECTOR_ADDR” 读取数据存入全局变量数组 DATA[128]。 ➢ LED 指示读写完毕→CPU 休眠→利用 CCS 查看 DATA[] 数据是否正确。
注意事项	例程代码仅支持 2G (含) 以下的 TF 卡。两个工程的区别是 13 章的工程带软件 SPI 协议, 可通过条件编译切换。

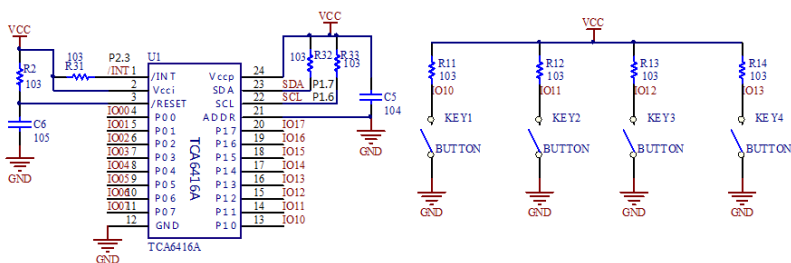
本例程主要学习 USCI_SPI 通信模块的配置及使用、软件 SPI 原理、SD 读写扇区操作。

- 1) 利用 SPI 协议收发数据的库函数位于 SPI.c 文件中, 库函数不针对特定外设, 只要是用 SPI 通信都适用。软硬件 SPI 用条件编译的方法进行了兼容。
- 2) SD 卡的操作时序是异常复杂的, 本例程只涉及利用 SPI 协议来控制 SD 卡, 就完成三件事: 初始化、写特定扇区和读特定性扇区。相关库函数可直接看 SD_SPI.h 头文件, 库函数不针对具体 IO 和硬件, 只要是读写 SD

卡都适用。

- 3) SD 卡与 SPI 协议之间的“接口”写在了 SD_HardWare 文件中，相当于是把 G2 的 SPI 和 SD 卡联系起来。
- 4) 在编写复杂外设程序的时候，需要遵循的原则是，先按外设说明书“捏造”出一堆空头支票函数，将说明书中的信息描述完。然后再去一一编写兑现“空头支票”的子函数。千万别想一步就把单片机的 IO 和外设时序联系起来，那样写程序是很恐怖的。

4.11 I2C 扩展 IO



8个控制LED，4个检测按键，4个控制LCD驱动器

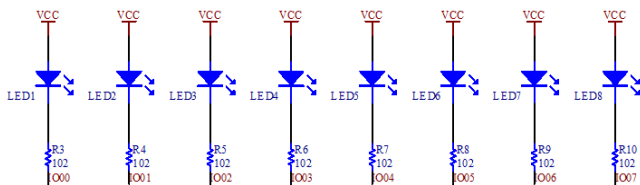


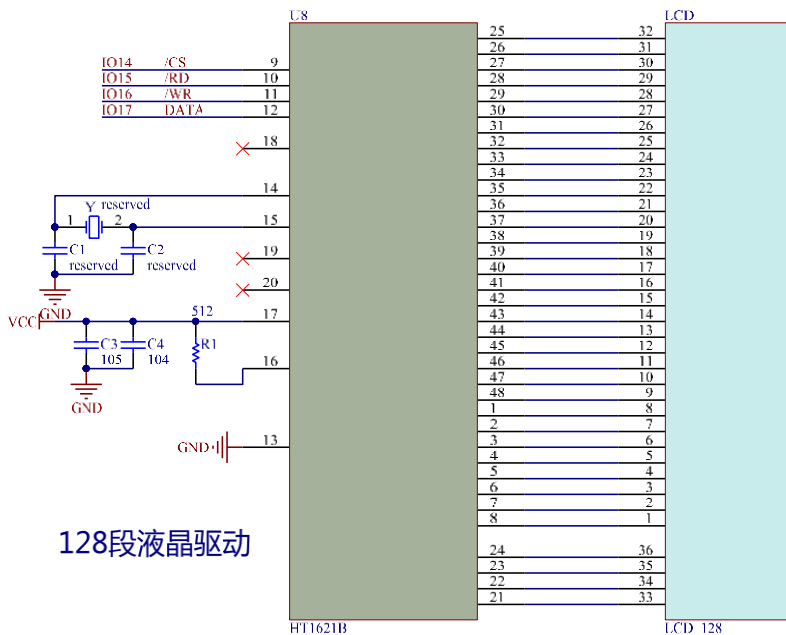
图 22 I2C 扩展 IO 实验原理图

工程名	12_I2C_LED_KEY; 13_2_LED_KEY_I2C_Hard_or_Soft
实验内容	通过 I2C 协议扩展出的 IO 识别机械按键和控制 LED
实验原理	<ul style="list-style-type: none"> ➢ 配置 DCO 频率→调用库函数 TCA6416A_Init() 实现 TCA6416A 初始化, 在 TCA6416A_Init()中包含了调用 I2C 的初始化函数 I2C_Init()。 ➢ 调用控制 I2C_IO 输出的函数 PinOUT(), 将 8 个 LED 设置为间隔亮灭。 ➢ 看门狗定时器设为 16ms 唤醒 CPU 一次, 然后执行 1 次 PinIN()函数以识别输入 IO 状态, 执行 1 次事件检测函数 I2C_IODect(), 在事件检测函数中, 包含有按键事件处理函数。 ➢ 事件处理函数的最终效果是 1 个按键切换 2 个 LED 的亮灭。
注意事项	G2 单片机的 I2C 口是 P1.6 和 P1.7, 一定要把 G2 板上 P1.6 的 LED 跳线拔掉, 否则 I2C 将无法通信。两个工程的区别在于有无软件 I2C 条件编译。

本例程主要学习 USCI_I2C 通信模块的配置及使用、软件 I2C 原理、TCA6416A 操作时序。

- 1) I2C 协议收发有关的库函数位于 I2C.c 中, 如果是初次学习 I2C 协议, 建议在 I2C.h 中条件编译, 使用软件 I2C, 这样可以比较清楚的知道协议内容。
- 2) TCA6416A 有关的操作函数位于 TCA6416A.c 文件中, 最核心的内容可以看 TCA6416A.h 中所列出的 3 个库函数 PinIN()、PinOUT()、TCA6416A_Init()和一个全局变

- 3) 调用一次 `PinIN()`函数,就意味着将 TCA6416A 的输入 IO 值写入全局变量 `TCA6416A_InputBuffer` 中。
- 4) 调用负责 `PinOUT()`函数则可控制任意扩展 IO 的输出。



工程名	14_LCD_SelfScan_Hard_or_Soft_I2C
实验内容	通过 I2C 协议扩展出的 IO 控制 HT1621 驱动 LCD 做自检显示
实验原理	<ul style="list-style-type: none"> ➤ 配置 DCO 频率→调用库函数 TCA6416A_Init() 实现 TCA6416A 初始化, 在 TCA6416A_Init() 中包含了调用 I2C 的初始化函数 I2C_Init()。 ➤ 分别调用 LCD_DisplaySeg()、LCD_DisplayDigit()、LCD_DisplayNum() 三种改写 LCD 显示缓存 LCD_Buffer[] 的程序, 改写想要实现的显示效果。 ➤ 调用 HT1621_Reflash() 函数更新显存至 HT1621 中, LCD 显示做相应变化。 ➤ LCD 先逐段显示, 再逐段消隐, 然后 8 字段显示 0-9 数字, 最后显示 PASS, 如图 24 所示。
注意事项	G2 单片机的 I2C 口是 P1.6 和 P1.7, 一定要把 G2 板上 P1.6 的 LED 跳线拔掉, 否则 I2C 将无法通信。

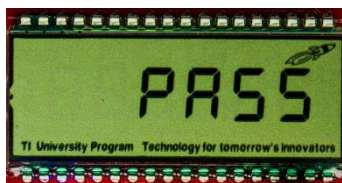


图 24 LCD 自检实验效果图

本例程主要学习 HT1621 驱动器的控制方法、显存隔离以及利用宏定义进行硬件隔离的编程思想。

- 1) 要想使用 MSP-EXP430G2 扩展板上这块 128 段式液晶比“登天”还要难。如图 25 所示, 单片机必须用 I2C 协

议去控制 TCA6416A 输出 4 个控制信号 CS、WR、RD、DATA，哪怕只是改变一次 CS 的电平，实际工作量就是完整的 I2C 通信了一次。

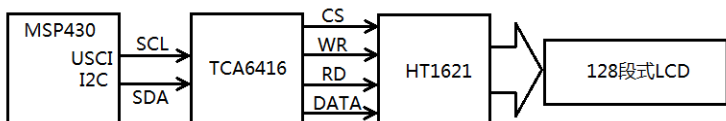


图 25 LCD 控制的信号控制图

- 2) 如图 26 所示，将 HT1621 的控制信号线宏定义之后，无论用的是何种 IO，HT1621 的库函数都是一样的。这就是利用宏定义进行硬件隔离的思想。

```

//-----控制信号线宏定义-----
#define HT1621_CS_LOW      PinOUT(14,0)
#define HT1621_CS_HIGH    PinOUT(14,1)
#define HT1621_RD_LOW     PinOUT(15,0)
#define HT1621_RD_HIGH    PinOUT(15,1)
#define HT1621_WR_LOW     PinOUT(16,0)
#define HT1621_WR_HIGH    PinOUT(16,1)
#define HT1621_DATA_LOW   PinOUT(17,0)
#define HT1621_DATA_HIGH  PinOUT(17,1)
  
```

图 26 利用宏定义进行硬件隔离

- 3) 对于显示类的应用，显存隔离也是一种常用的编程思想。在 RAM 中建立显存数组 LCD_Buffer[]，CPU 想显示什么内容就调用函数改写显存数组，这是顶层程序要干的事。

- 4) 至于显存数组中的数据怎么写到 HT1621 硬件中, 那是 HT1621_Reflash()函数要实现的事, 这属于硬件相关的底层程序。

4.13 拨盘电位器

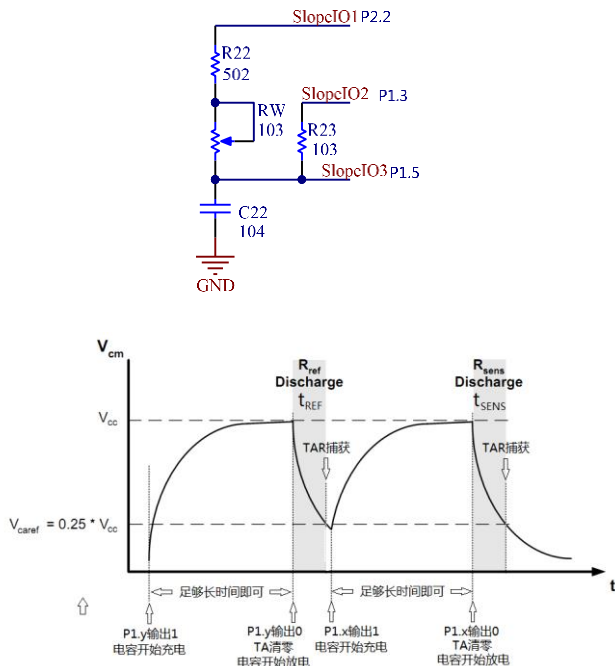


图 27 拨盘电位器实验原理图

工程名	17_Slope_ADC
实验内容	通过 RC 充放电计时，间接计算待测拨盘电位器电阻 RW 的大小，并根据阻值控制 LED 灯柱长度和 LCD 的显示内容。
实验原理	<ul style="list-style-type: none"> ➢ 配置 DCO 频率→调用库函数 TCA6416A_Init() 实现 TCA6416A 初始化，在 TCA6416A_Init() 中包含了调用 I2C 的初始化函数 I2C_Init()。 ➢ 调用 HT1621_init() 函数实现 LCD 显示初始化，调用 Display_SLOPE() 让显存写入“SLOPE”和符号等固定不变的字样，调用 HT1621_Reflash() 函数刷新显存数据到 HT1621 中，LCD 显示“SLOPE”。 ➢ WDT 设为 16ms 中断，中断子函数内完成 RC 充放电，测量阻值的程序，并调用事件处理函数。 ➢ 事件处理函数有阻值显示函数 Display_RSNS()，LED 灯柱控制函数 TCA6416A_LED()。
注意事项	G2 单片机的 I2C 口是 P1.6 和 P1.7，一定要把 G2 板上 P1.6 的 LED 跳线拔掉，否则 I2C 将无法通信。

本例程主要学习 SLOPE 型 ADC 的原理以及管理多个库函数文件。

- 1) 如图 27 所示的充放电波形为 SLOPE ADC 的基本原理。充电足够长的时间后，SlopeIO 置低开始放电，放电至 0.25VCC 时，触发比较器中断，通过读取 TAR 就可以知道放电时间。放电时间与放电电阻的大小成正比，从而可计算出待测电阻阻值。

- 2) 凡是要使用 LCD 显示，均需包含 HT1621.c、I2C.c、LCD_128.c、TCA6416A.c 四个库函数文件及其头文件，这些文件统一放在 TCA6416A 文件夹中。
- 3) Timer_A3.c 和 ComprartorA.c 文件是 TA 和比较器模块利用 Grace 生成的初始化代码的移植。
- 4) Comparator.c 文件是 SLOPE ADC 核心的测频（测电阻）相关函数。

4.14 温度传感器采样及显示

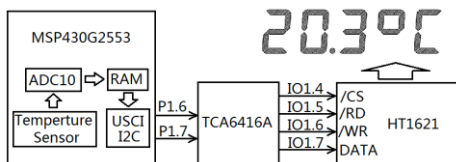


图 28 温度传感器采样及显示例程原理图

工程名	19_ADC10_Temperature
实验内容	通过移植 CCS Example 中的测温代码，将温度值显示到 LCD 屏幕上。
实验原理	<ul style="list-style-type: none"> ➤ CCS 的 Example 有一个 ADC10 TempSens Convert 的例程，将测温代码进行移植，ADC 采样值存入 long 型全局变量 temp，并换算为摄氏度，存入 long 型全局变量 IntDeg。 ➤ 对 IntDeg 进行数字拆分，并显示在 LCD 屏幕上。

注意 事项	G2 单片机的 I2C 口是 P1.6 和 P1.7，一定要把 G2 板上 P1.6 的 LED 跳线拔掉，否则 I2C 将无法通信。
------------------	---

本例程主要学习如何借助 CCS 的 Example 加快程序开发进程。

- 1) 对于大多数外设，CCS 的 Example 中都给出了参考例程，如图 29 所示。移植例程中的代码可以显著加快学习和使用该外设的速度。
- 2) ADC10 TempSens Convert 例程功能是对 G2 单片机片内温度传感器进行采样。我们直接读取采样结果，然后显示在 LCD 上。
- 3) 不要忘记添加与 LCD 显示相关的 4 个 c 文件及其头文件。

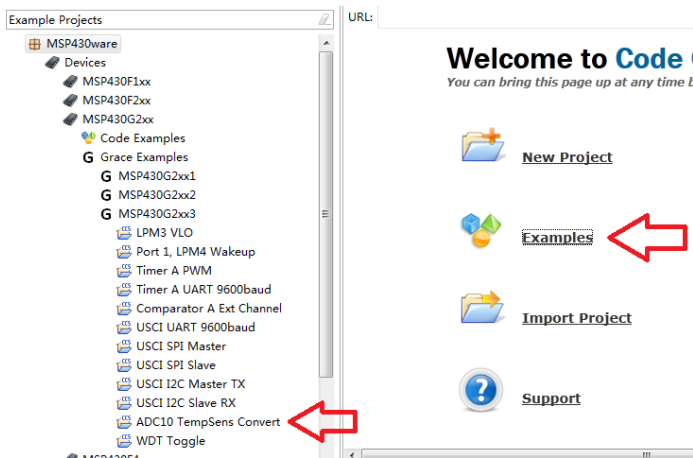


图 29 CCS 中的 Example

4.15 SPWM 波形合成及采样

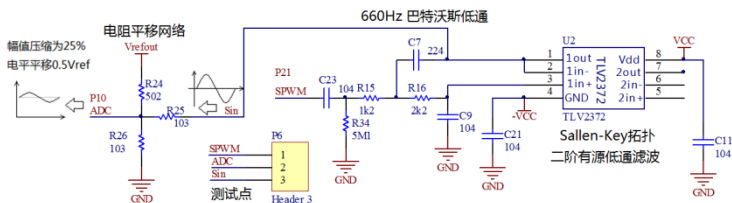


图 30 SPWM 波形合成及采样原理图

工程名	20_Sin_Gen_and_Sample
实验内容	通过 TA_PWM 输出占空比按正弦表变化的 PWM 波形。PWM 经有源滤波器变为正弦波后再由 ADC10 采样。
实验原理	<ul style="list-style-type: none"> ➤ TA 可以很方便的输出 PWM 波形，如果周期性的按“正弦表”改变 PWM 的占空比，就可以得到所谓的 SPWM 波形。 ➤ SPWM 经电容隔直，低通滤波后，可以变成交流正弦波。 ➤ 交流正弦波经 3 电阻网络变成 0.25 倍，平移 1.25V 的单极性正弦波并进入 ADC10 采样。 ➤ 可以用示波器观察交流正弦信号以及缩放平移后的单极性正弦信号。也可以用仿真软件观察 ADC10 采样的单极性正弦信号。
注意事项	请勿使用 CCS 的 graph 功能观测 ADC10 采样信号，因为低通滤波器需要一定时间才能输出正常的滤波效果，断点引入会导致结果失真。

本例程主要学习利用正弦表生成 SPWM 的原理，以及三电阻网络平移缩放信号的原理。

- 1) 查表法是合成 SPWM 信号的主要方法。按正弦表周期性改变 TA_PWM 的占空比就可以得到 SPWM。
- 2) SPWM 经过低通滤波后，可以得到和正弦表一致的正弦信号。
- 3) 如图 30 所示的三电阻网络可以对信号进行缩放和平移，具体参数可以通过叠加原理进行计算。
- 4) 示波器观测的 SPWM 滤波后和缩放平移后的信号如图 31 所示。

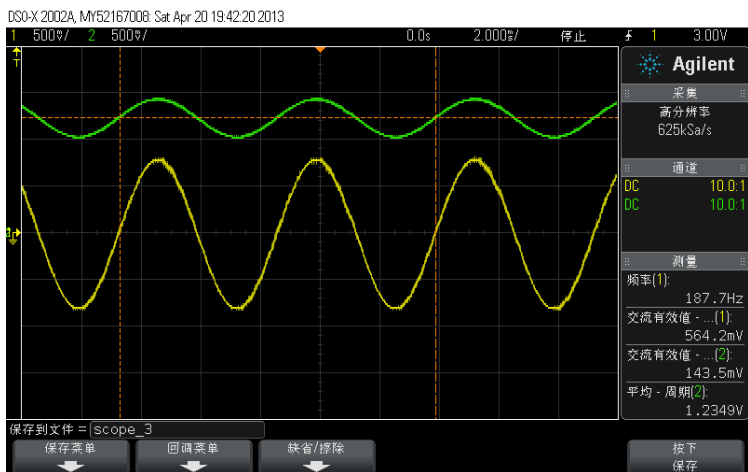


图 31 示波器观测到的 SPWM 滤波后的信号

- 5) 除了示波器直接观测信号外，如图 32 所示，可以用仿真器在 CCS 中将 ADC10 采样数据复制出来，再用 EXCEL 图表功能也能还原出正弦波（图 33）。

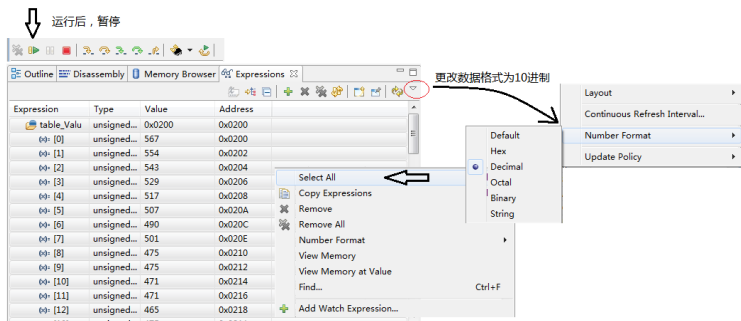


图 32 利用 CCS 查看变量数组的数据

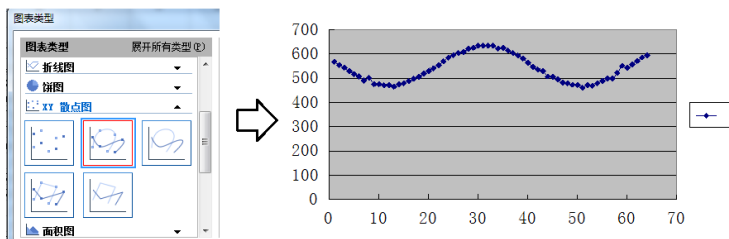


图 33 利用 EXCEL 图表功能还原信号

4.16 任意波形发生器 AWG

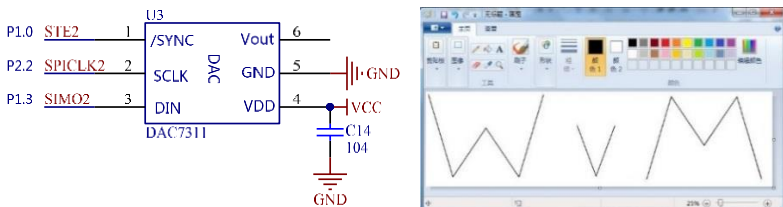


图 34 任意波形发生器原理图

工程名	21_1_DAC_AWG
实验内容	生成“WvM”波形的“正弦表”。周期性调用 DAC 依次输出“正弦表”数据，就构成了任意波形发生器 AWG。
实验原理	<ul style="list-style-type: none"> ➤ 如图 34 所示，通过画图板画出 WvM 形状图形，利用波形采点软件提取图形的坐标，生成“正弦表”。 ➤ 定时调用 write2DAC8411()函数，将正弦表中数据依次作为模拟量输出。 ➤ 用示波器观察 DAC 输出信号是否为 WvM，如图 35 所示。
注意事项	DAC8411、DAC8311、DAC7311 为系列串行 DAC 产品，位数不同，但是编程代码兼容按 DAC8411 编写的代码。即替换不同芯片，代码可以完全不改动，只不过低位数据无效而已。

DSO-X 2002A, MY52167008 Mon Apr 22 23:05:59 2013

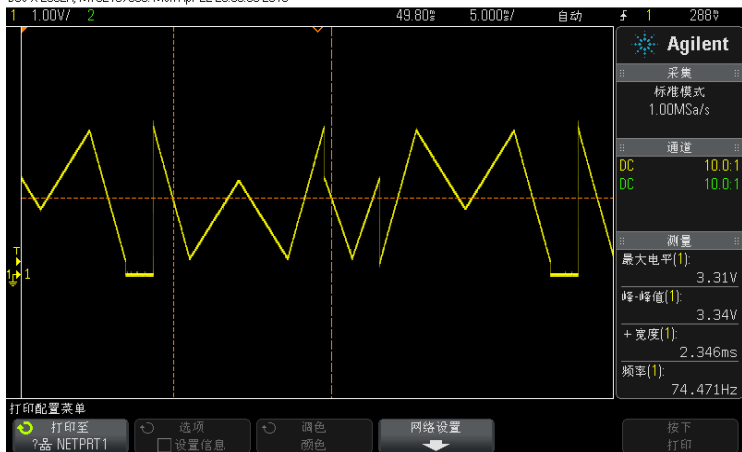


图 35 AWG 示波器观测效果

本例程主要学习任意波形发生器 AWG 的原理，以及系列串行 DAC 芯片的控制方法。

- 1) AWG 以及 DDS 的原理都是用 DAC 周期性按“数据表”输出信号，这些数据表描述的形状就是 DAC 输出波形的形状。
- 2) DAC8411、DAC8311、DAC7311 分别为 16、14、12 位串行 DAC，使用 3 个普通 GPIO 控制。针对 DAC8411 的代码完全兼容其他两个 DAC。
- 3) 为了精确还原“WvM”波形，图形量化点的数目取了

4.17 基于 AWG 的音频播放器

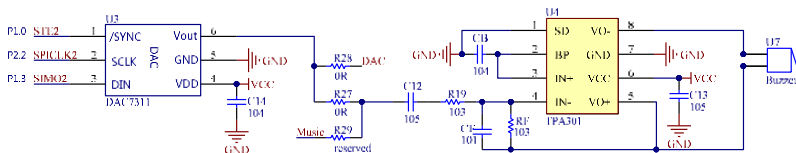


图 36 音频播放器原理图

工程名	21_2_TF_Audio
实验内容	CPU 读取 TF 卡中的 wav 格式音频数据，DAC 还原数据并经音频放大器驱动蜂鸣器播放音乐。
实验原理	<ul style="list-style-type: none"> ➤ wav 格式的音频是未经压缩的，所以可以直接读出，用 DAC 还原出音频。 ➤ 将 wav 格式歌曲存放在 TF 卡中，利用 winhex 软件获取歌曲文件的物理扇区地址。 ➤ 从 TF 卡指定扇区读取音频数据，再用 DAC 还原，音频功放放大后就可以由蜂鸣器播放音乐了。
注意事项	为配合例程，TF 卡中音频应为单声道，采样率 11025Hz。

本例程主要学习音频数据的 DAC 还原方法，以及实际应用 TF 卡作为大容量存储介质的方法。

- 1) 所有与操作 SD 卡 (TF 卡) 有关的文件都必须使用。
- 2) DAC 还原模拟信号的“速度”与音频文件的采样率必须一

致，这样音乐才不会变调。

3) G2 单片机的 RAM 太小，不能使用文件系统的方法来管理 SD 卡，只能按读写扇区的原始方法读写 SD 卡数据，这样一来就需要借助 winhex 软件才能知道音频文件存放的物理扇区地址（如图 37）。

4) 范例音乐文件在序言页的技术支持网站 www.hpati.com 下载，TF 卡请自备（需小于等于 2G）。

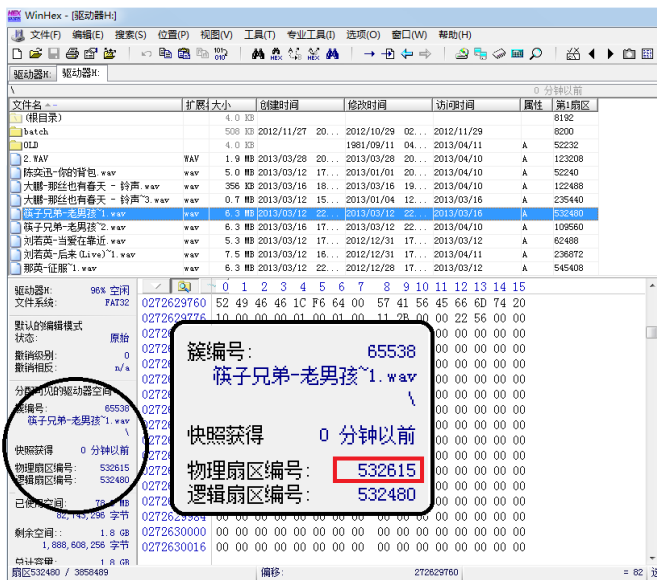


图 37 查看文件的物理扇区地址

4.18 自校验 DCO

工程名	22_DCO_Calb_Test
实验内容	自校验 DCO 1-16MHz 共 16 个频率的最佳配置参数，存入 InfoFlashB 中，并周期性输出这些 DCO 频率供示波器观测和频率计检测。
实验原理	<ul style="list-style-type: none"> ➤ 通过外接 32.768kHz 晶振，可以有办法测量出 DCO 当前的准确频率值。 ➤ 将 DCO 配置 RSELx 和 DCOx 得到的 128 个频率都测量出来。 ➤ 找出最接近目标频率的 RSELx 和 DCOx 参数，再改变 MODx 测试一遍，找到最合适的 MODx 参数。 ➤ 将 16 组三参数都存入 InfoFlash 中，供 DCO 配置调用。 ➤ 校验完毕后，间隔 1 秒依次将 DCO 配置为 1-16MHz，并通过 P1.4 口观测 DCO 频率。
注意事项	为了不影响原厂存在 InfoFlashA 段的 DCO 校验参数，例程中自校验参数是写入 InfoFlashB 段。但是 CCS 工程默认的 debug 下载程序时，会把除了 A 段以外的全部 flash 擦除。为了让自校验的参数得以保留（不会因每次下载代码被擦除），应该如图 38 所示，将 Debug 选项改为“Erase main memory only”。

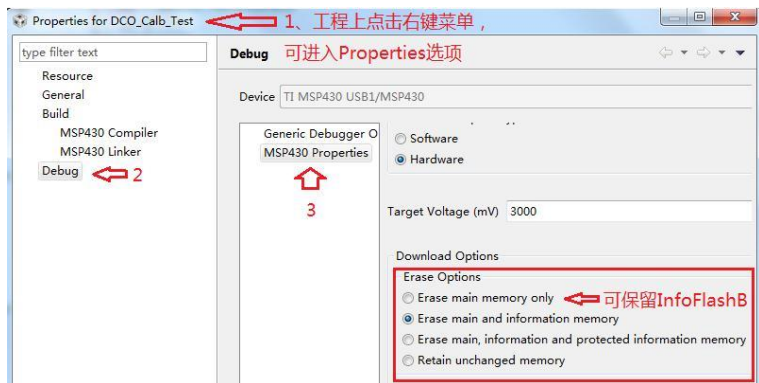


图 38 CCS 工程默认的擦写选项

本例程主要学习利用 1 个准确时钟，测量其他时钟频率的原理。调用 flash 控制器读写非掉电易失数据的方法。

- 1) 如图 39 所示，WDT 中断使用精确的 32.768kHz 晶振作为时钟源，所以其 16ms 定时是准确的。
- 2) TA 的时钟源选择为 DCO，测量 16ms 间隔 TAR 的计数值，就可以计算 DCO 的准确频率。
- 3) 图 39 中的数据显示，DCO 的准确频率为 $25535/16\text{ms}=1.6\text{MHz}$ 。

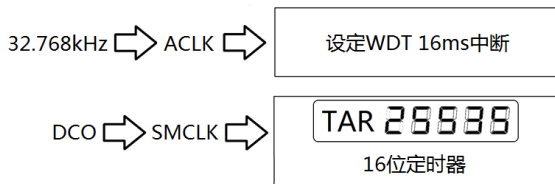


图 39 测频原理

- 4) Flash 控制器是全系列 MSP430 标配的片内外设，这意味着无需外扩存储器就可以实现掉电不失存储。读写 Flash 的库函数在 Flash.c 文件中。